Making secure software

How should we write software that will be secure?

- Flawed approach: Design and build software, and ignore security at first
  - Add security once the functional requirements are satisfied!
  - The software may has important vulnerability at the end

- Better approach: Build security in from the start
  - Incorporate security-minded thinking into all phases of the development process
  - Can avoid missing important security requirements

Development process

Many development processes; **four common** phases:

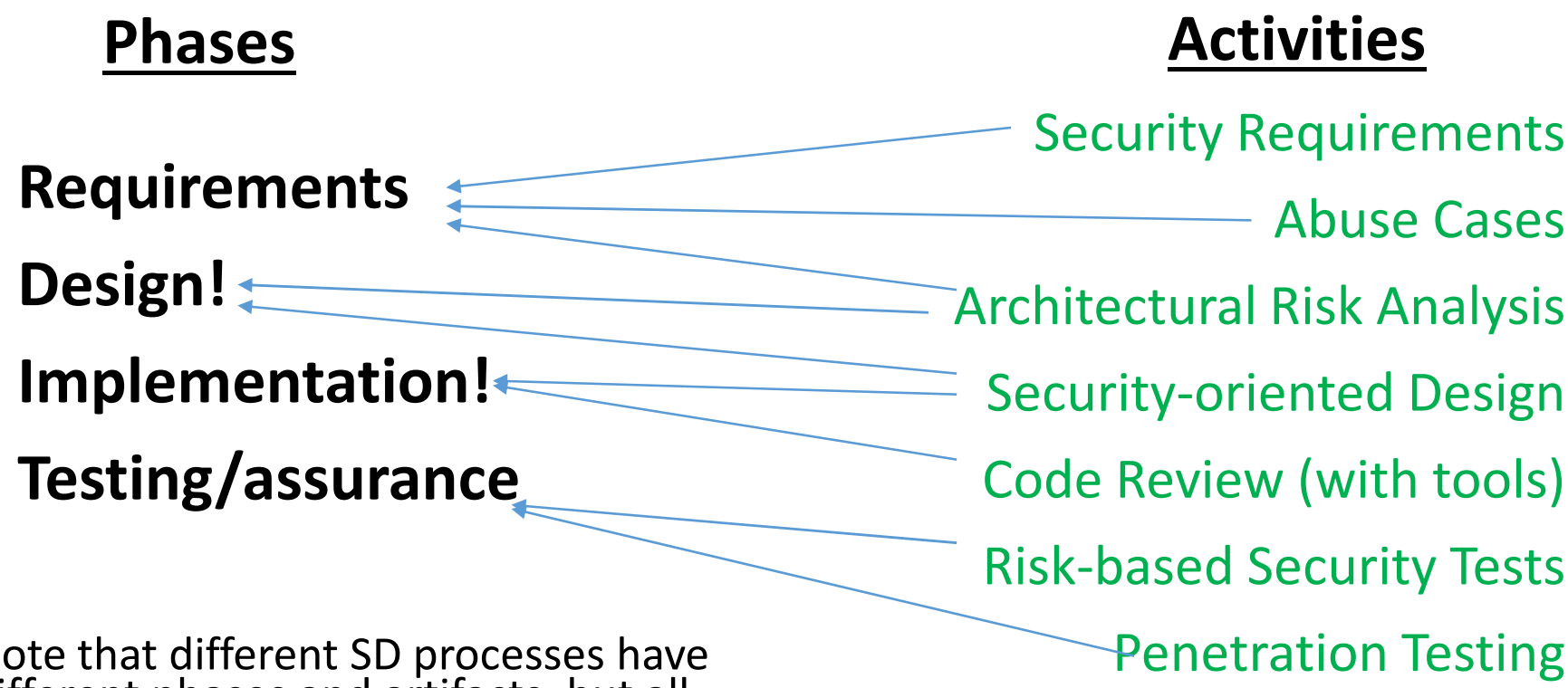
1. **Requirements**! Involves determining what the software should do, and not
2. **Design**! looks at how to structure the system to meet these requirements
3. **Implementation**! Involves actually writing code to implement the design
4. **Testing/assurance**! Involves checking that the implementation actually doe what it's supposed to

Phases of development apply to the whole project, its individual components, and its refinements/iterations!

Where does security engineering fit in?

- **All phases!**

Security engineering

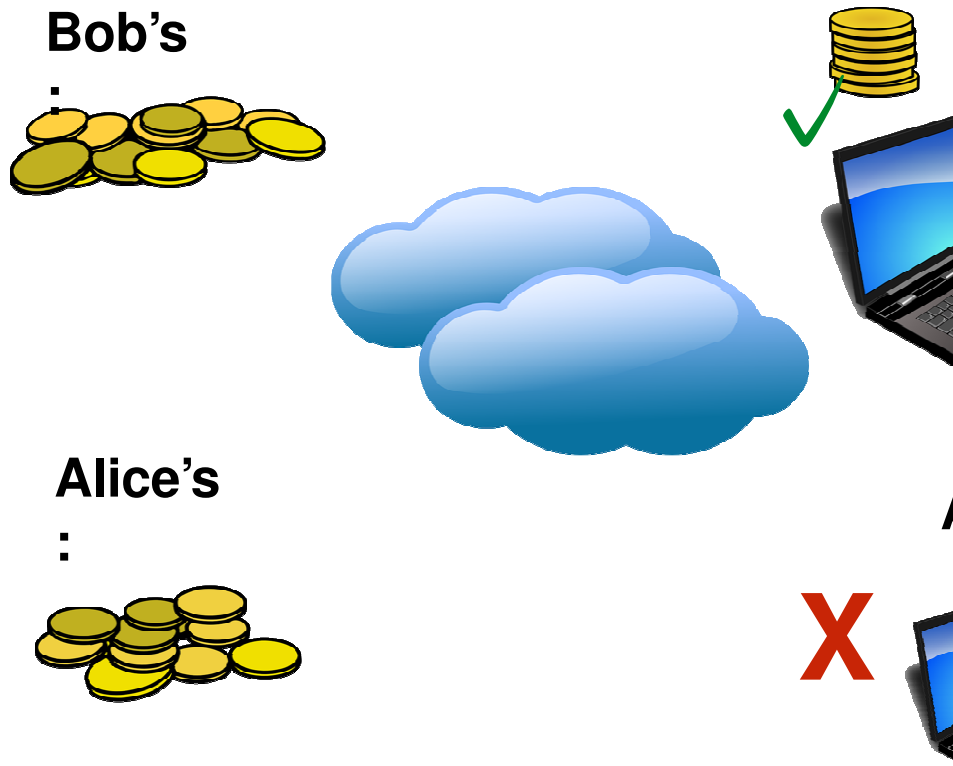


ote that different SD processes have
ifferent phases and artifacts, but all
nvolve the basics above. We'll keep it
mple and refer to these.

Running Example: On-line banking

What requirements might we have for this application?

- Account holders be able to deposit funds into their accounts
- Prevent other people who are not authorized by the account holder from withdrawing those funds
- The account holder accesses their funds whenever time they choose



Threat Modeling

The **threat model makes explicit the adversary's assumed powers!**

- Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong

The threat model is **critically important**

- If you are not explicit about what the attacker can do, how can you assess whether your design will repel that attacker?

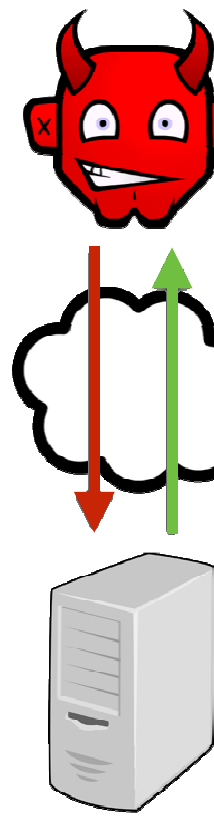
- This is part of **architectural risk analysis**

Example: Network User

An (anonymous) user that can connect to a service via the network

Can:

- **measure** the size and timing of requests and responses
- run **parallel sessions**!
- provide **malformed inputs, malformed messages**
- **drop or send extra messages**

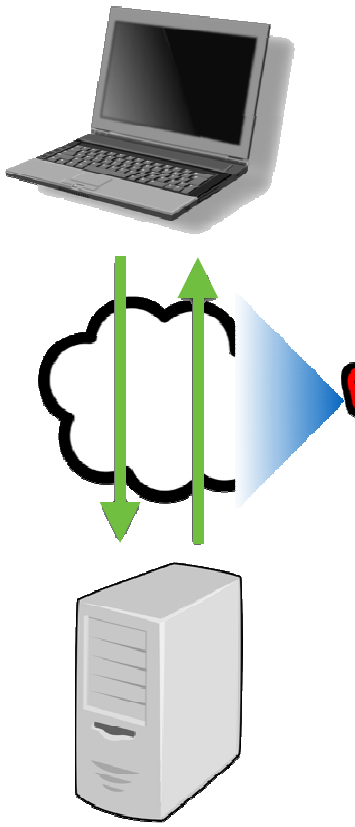


- Example attacks: SQL injection, XSS, CSRF, buffer overrun/ROP payloads, …

Example: Snooping User

Internet user **on the same network** as other users of some service

- For example, someone connected to an unencrypted Wi-Fi network at a coffee shop



Thus, can additionally

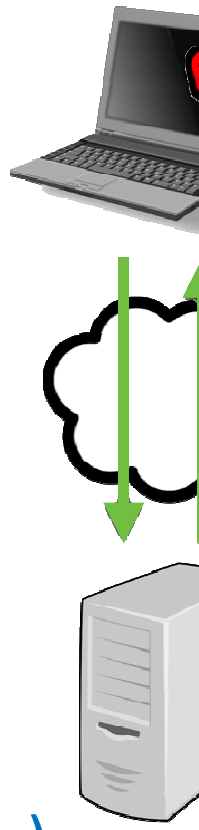
- **Read/measure** others' **messages**,
- **Intercept, duplicate,** and **modify messages!**

Example attacks: **Session hijacking** (and other data theft), **privacy-violating side-channel attack**, **denial of service**

Example: Co-located User

Internet user **on the same machine** as other users of some service

- E.g., malware installed on a user's laptop



Thus, can additionally

- **Read/write** user's **files** (e.g., cookies) and memory
- Snoop keypresses and other events
- Read/write the user's **display** (e.g., to spoof)

Example attacks: Password theft (and other credentials/secrets)

Threat-driven Design

Different threat models will elicit different responses

Network-only attackers implies **message** traffic **is safe**

- No need to encrypt communications
- This is what *telnet* remote login software assumed

Snooping attackers means **message** traffic **is visible**

- So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)
  - Which is most appropriate for your system?

Co-located attacker can **access local files, memory**

- Cannot store unencrypted secrets, like passwords

Bad Model = Bad Security

Any assumptions you make in your model are potential **holes that the adversary can exploit!**

E.g.: Assuming no snooping users **no longer valid**

- Prevalence of wi-fi networks in most deployments

Other mistaken assumptions

- **Assumption**: Encrypted traffic carries no information
  - Not true! By analyzing the size and distribution of messages, you can infer application state
- **Assumption**: Timing channels carry little information
  - Not true! Timing measurements of previous RSA implementations could be used eventually reveal a remote SSL secret key

Finding a good model

Compare against similar systems

- What attacks does their design contend with?

Understand past attacks and attack patterns

- How do they apply to your system?

Challenge assumptions in your design

- What happens if an assumption is untrue?
  - What would a breach potentially cost you?
- How hard would it be to get rid of an assumption, allowing for a stronger adversary?
  - What would that development cost?

Security Requirements

Security Requirements

Software requirements typically about **what the software should do**

Security requirements Security-related goals (or policies)

- **Example**: One user's bank account balance should not be learned by, or modified by, another user, unless authorized

Required mechanisms for enforcing them

- **Example**:
  - Users identify themselves using passwords,
  - Passwords must be "strong," and
  - The password database is only accessible to login program.

Typical *Kinds* of Requirements

Policies

- **Confidentiality** (and Privacy and Anonymity)
- **Integrity**
- **Availability**

Supporting **mechanisms**

- **Authentication**
- **Authorization**
- **Auditability**

Privacy and Confidentiality

Definition: **Sensitive information not leaked** to unauthorized parties

- Called *privacy* for individuals, *confidentiality* for data
- **Example** policy: bank account status (including balance) known only to the account owner

Leaking **directly** or via **side channels!**

- **Example**: manipulating the system to directly display Bob's bank balance to Alice
- **Example**: determining Bob has an account at Bank A according to shorter delay on login failure

Secrecy vs. **Privacy**?

Anonymity

A specific **kind of privacy**

Example: Non-account holders should be able to browse the bank informational site without being tracked

- Here *the adversary is the bank*
- The previous examples considered other account holders as possible adversaries

ntegrity

Definition: **Sensitive information not damaged** by (computations acting on behalf of) unauthorized parties

Example: Only the account owner can authorize withdrawals from her account

Violations of integrity can also be **direct** or **indirect**

- **Example**: Being able specifically withdraw from the account vs. confusing the system into doing it
  - For example, by using a cross-site request forgery (CSRF).

Availability

Definition: A system is **responsive to requests**

Example: a user may always access her account for balance queries o
withdrawals

Denial of Service (DoS) attacks attempt to **compromise availability** b
busying a system with useless work or cutting off network access

Supporting mechanisms

Leslie Lamport's defines **gold standard** mechanisms provided by a system to enforce its requirements

- **Au**thentication
- **Au**thorization
- **Au**dit

The gold standard is **both requirement and design**

- The *sorts of policies* that are authorized *determines* the *authorization mechanism*
- The *sorts of users* a system has *determines* how they should be *authenticated*

Authentication

What is the **subject of security policies**?

- Need to define a ***notion of identity*** and a way to ***connect an action with an identity!***
  - By who we mean a **principal.** That is, it could also be a human being. Or it could be some service or a computer program.

How can system tell a user is who he says he is?

- What (only) he **knows** (e.g., password)
- What he **is** (e.g., biometric)
- What he **has** (e.g., smartphone)
- Authentication mechanisms that employ more than one of these factors are called **multi-factor authentication**
  - E.g., bank may employ passwords and text of a special code to a user's smart phone

Authorization

Defines **when a principal may perform an action!**

Example: Bob is authorized to access his own account, but not Alice's account

There are a wide variety of **policies** that define what actions might be authorized

- E.g., access control policies, which could be
  - originator based: the users performing particular actions , like Alice or Bob.
  - role-based: a bank teller or a bank manager may be allowed to perform certain things.
  - user-based: Bob could originate a policy that allows Alice to withdraw a fixed amount of money from his account.
  - etc.

Audit

Retain enough information to be able to **determine the circumstances of a breach or misbehavior** (or*establish one did not occur*)

Such information, often stored in **log files**, must be **protected from tampering**, and from access that might violate other policies

Example: Every account-related action is logged locally and mirrored at a separate site

Defining Security Requirements

Many processes for deciding security requirements

Example: **General policy concerns**

- Due to **regulations**/standards
  - E.g. HIPAA: Health Insurance Portability and Accountability Act in US
- Due **organizational values** (e.g., valuing privacy)

Example: **Policy arising from threat modeling**

- Which **attacks** cause the **greatest concern**?
  - Who are the likely adversaries and what are their goals and methods?
- Which **attacks** have **already occurred**?
  - Within the organization, or elsewhere on related systems?

Abuse Cases

Abuse cases illustrate security requirements

Where use cases describe what a system *should* do, **abuse cases describe what it should *not* do!**

Example **use case**: The system allows bank managers to modify an account's interest rate

Example **abuse case**: A user is able to spoof being a manager and thereby change the interest rate on an account

Defining Abuse Cases

Using attack patterns and likely scenarios, construct cases in which ar
adversary's exercise of power could **violate a security requirement**

- Based on the threat model
- What might occur if a security measure was removed?

Example: *Co-located attacker* steals password file and learns all user passwords

- Possible if password file is not encrypted

Example: *Snooping attacker* replays a captured message, effecting a bank withdrawal

- Possible if messages are have no *nonce*